Robotics Practicals

Topic 2: Teaching Robots to Acomplish a Manipulation Task

Jan FROGG, Sascha FREY, Stanislas FURRER

April 8, 2019

1 Introduction

The purpose of this robotics practical is to teach a robot to recognize, locate, grab and stack objects of different shapes and sizes in order to build a tower. For this purpose, we use already implemented programs on the Robot Operating System (ROS) framework and MATLAB to analyze the data.

The first part of the practical consists of the identification of the different objects, with the offline implementation of principal component analysis (PCA) algorithm. The aim of this step is to compress the collected images in order to save memory and be able to classify them into the correct object classes.

The second step is the estimation of the localisation of the objects with respect to the robot frame. For this purpose, we calibrate the robot camera and the workspace camera.

Finally we teach, learn and generate the appropriate motion to pick up each object and put it on top of other objects, by using an offline Gaussian mixture regression (GMR) algorithm. The robot used is a light-weight, small-size, six degrees of freedom (DOF) Katana arm, developed by Neuronics.

2 Object recognition

The goal of the first session was to build a PCA model. We defined three object classes to build our tower. In order to minimize misclassification we have taken care to take objects of very different colors and shapes that can be stacked easily. The specifications of the blocks are further developed in Figure 1.

Explain how you created your training set: How many and what types of images did you pick for each object? Justify your choices.

In the final implementation of this practical the objects would randomly be placed on the workspace. Hence the classification step should be robust for all possible orientations, lighting conditions and positions of the object in the workspace. We estimated that 30 pictures of each object would be sufficient for the algorithm to generalize orientation and brightness. In the 30 pictures we count ; 10 with different angles, 10 with different brightness and angles and 10 with different positions, angles and brightness. Figure 2 shows samples from class 1. The picture is already processed (i.e. converted to gray-scale and cropped).

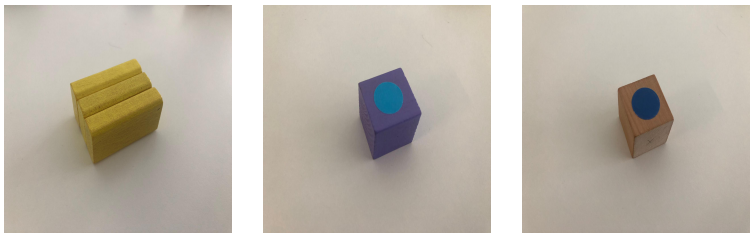


Figure 1: The three object: (***Left***) class 0, dim 3.7x5x2.4 cm^3, (***middle***) class 1, dim 2.8x2.8x2.8 cm^3, (***right***) class 2, dim 2.8x2.8x2.8 cm^3

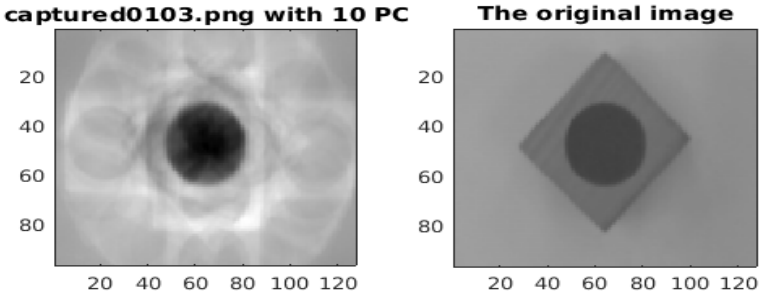


Figure 2: Object before and after PCA

Validate your trained model with some new images (i.e. Step 7). How many and what type of test images did you use to have a good validation of the trained model? Justify your choices. What is the performance of your object recognition model?

To validate our model, we chose images obtained in the same way as the the training ones (different angles, brightness and position). We chose to take 5 pictures of each class as the testing set. This represents a 20 % testing/training ratio. This choice appears to be good as can be seen in Figure 3. In fact, we can see that the performance of our PCA is optimal if the number of principal components is higher than 6. This is therefore the number that we use to proceed.

Is there any misclassication? If yes, explain the source(s) of misclassication? If no, explain what experiment conditions you have considered to obtain this performance? Could you think of other means to validate your model?

There was only one misclassification due to a similarity between two objects. A way to increase the robustness of our model would be to add pictures of the blocks while they were slightly out of the frame in the training set. Indeed, this can happen during the final test. This is largely due to the cropping process of the image.

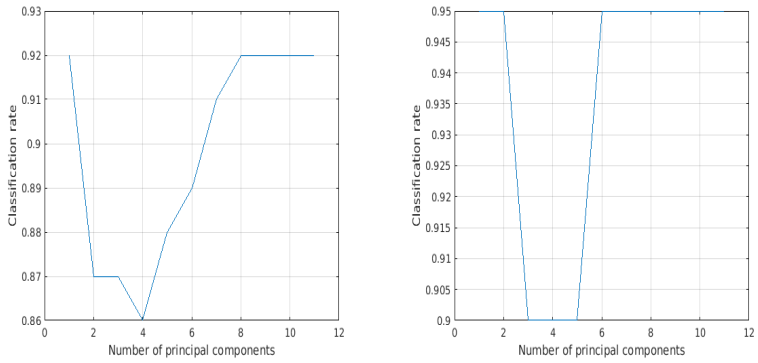


Figure 3: Training (left) and testing (right) sets accuracy as a function of the number of PCs used.

How many bytes are used to store these images for each object in MATLAB before and after PCA (you need to understand the example mentioned in Section 2.2.2)? What is the optimal number of eigenvectors for your dataset and how did you pick it?

As previously discussed, we chose to keep 6 eigenvectors for our model because it was the best trade off between training and testing accuracy.

Moreover, the purpose of running PCA on our dataset is not only to project our data into a space that is easily separable by a classifier, but also to save memory when storing the pictures. Since storing the pictures might require to decompress them afterwards, we need to keep a meaningful amount of data of the pictures so that the reconstruction is usable. For data reconstruction it is common to keep enough eigenvectors to explain 90% of the variance.

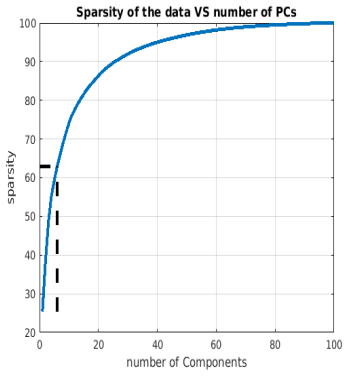


Figure 4: Percentage of the data's variance as a function of 6 principal components kept.

We can see in Figure 4 that by keeping 6 principal components, we explain 63% of the variance. This is not a good trade off between the complexity of the model, the training and testing accuracies and the information kept for data reconstruction. If the aim of conducting PCA was to compress the image, one should probably take more principal components. Before the PCA, we need to store N = Image of width W=96 pixels, heigth H =128 pixels in gray scale in an integer operator. Therefore,

before PCA the size of the memory needed is:

$$Memory = (N \cdot W \cdot H) \cdot sizeof(int) = (90 \cdot 96 \cdot 128) \cdot sizeof(int)$$

After PCA, each object will only have 6 coordinates. We will then have to store 6 coordinates (one for each eigenvector) for each object plus the six eigenvectors, which are also W*H dimensional images. The memory needed now is:

$$Memory = (6N + 6WH) \cdot sizeof(int) = (6 \cdot 90 + 6 \cdot 96 \cdot 128) \cdot sizeof(int)$$

Reducing the dimensionality of the dataset using PCA saved us then around 80% memory

3 Calibration

Explain why the robot and the camera are needed to be calibrated.

In order to effectively use the data from the camera, it is important to know the relative position and orientation between the camera and the robot. In order for the robot to grab an object that was located by the camera, we need to compute the equation system that will translate the location found by the camera into the robot's coordinate system.

Explain how you created the training set: how many points did you pick? Justify your choices.

In order to calibrate, we chose four points within the robot's working space. Three points are required to define a plane, choosing the additional point provides more accurate calibration.

What is the minimum number of the points you need to calibrate cameras? Is it enough? Justify your answer.

When defining the coordinates, only x and y mattered. To define a plane, one needs at least 3 points for the calibration. We used more to get more precision and to try and minimize the error.

Validate your trained model with some new datapoints. How many points did you use to have a good validation of the trained model? What is the performance of your calibration? How can you improve it?

In order to validate our model, we used 3 points. This gives us a training / testing ratio of nearly 60%. We obtained a training error of $2.7326 \times 10^{-10} mm$, and a testing error of $0.3400 mm$. This is well within the required tolerance for the application we are considering. There might be many reasons that could explain this difference in performance. The first one is the manual setting of the correspondence between the camera positions and the robot positions for the training. Indeed, the camera positions were generated by automatically detecting an object in the camera frame. Then, the robot positions were generated by moving the robot's end effector to the object location. One way to improve the performance could have been to pay thorough attention when moving the robot to try to be as precise as possible. Secondly, we might not have recorded enough datapoints for the training part. Even though we recorded more points than the minimum required, with only 4 datapoints the model might be too well adjusted to the training data, hence adjusting to the errors that we made when saving the positions in the robot and camera frame. To sum up, performance may have been better if we were more careful when recording the positions and if we had more point correspondences recorded by different members of our group in order to eliminate the bias of one member moving the end effector in one specific way.

4 Teaching a reaching movement to a robot

4.1 Exercise 3

In this exercise we use a number of trajectories that have been shown to the robot in order to create a motion model for the trajectories of picking and placing objects. The recorded and stored trajectories are shown on Figure 5.

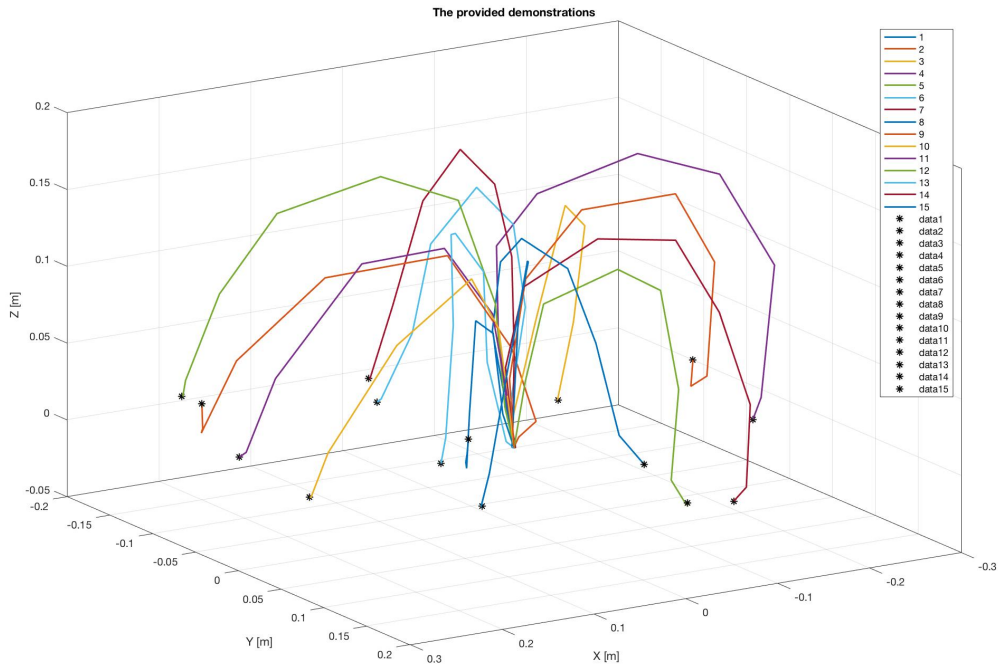


Figure 5: Plot of the various trajectories generated by SEDS for picking.

We note here that although the instructions called for separate demonstrations of picking and placing, we ultimately only created one training set for both motion types due to them being extremely similar and being under time constraints. The time limitations were largely due to an original set of demonstrations giving poor results. This was because of a general skew in the demonstration data, as seen in Figure 6.

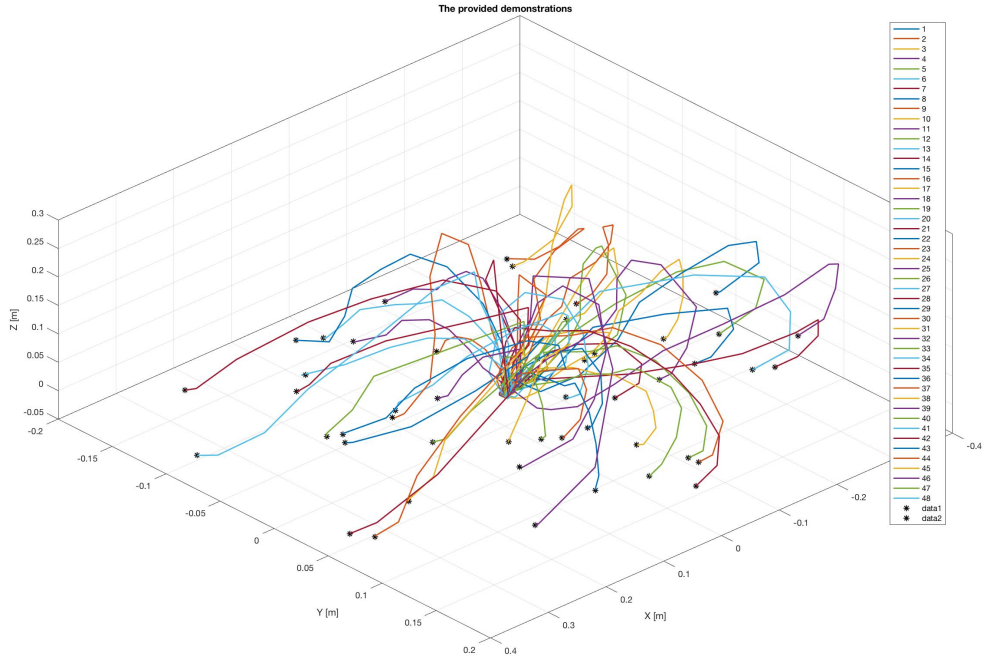


Figure 6: Plot of the various trajectories with a general skew induced in the data.

Next, we run the SEDS package with different parameters. In particular, we vary the number of gaussians between 1 and 6, and we change the optimization criteria between the mean squared error (MSE) and the Likelihood. We repeat a run of the algorithm for each state three times to account for the randomness in initialization and retain the lowest value. Results are listed in Table 1.

Number of gaussians	**MSE as criterion**	**Likelihood as criterion**
1	0.00233672	-9.9705118
2	0.00150397	-10.448409
3	0.00124836	-12.151494
4	0.00105848	-12.690056
5	0.00082924	-13.316988
6	0.00081108	-13.853107

Table 1: Values for MSE as well as for Likelihood as a function of the number of gaussians used for the trajectories in SEDS.

The effect of increasing the number of gaussians can clearly be observed by looking at the measure of error. Generally, the error becomes smaller, because more gaussians can account for greater variability in the data and thus create more complex models. As an example we can look at the case with only two gaussians and using the MSE as the criterion. Visible on Figure 7 we identify a model that is insufficiently fitted, most visible in the approach to the target point which is from

the negative direction in ξ_3. This observation implies a trajectory that goes through the table.

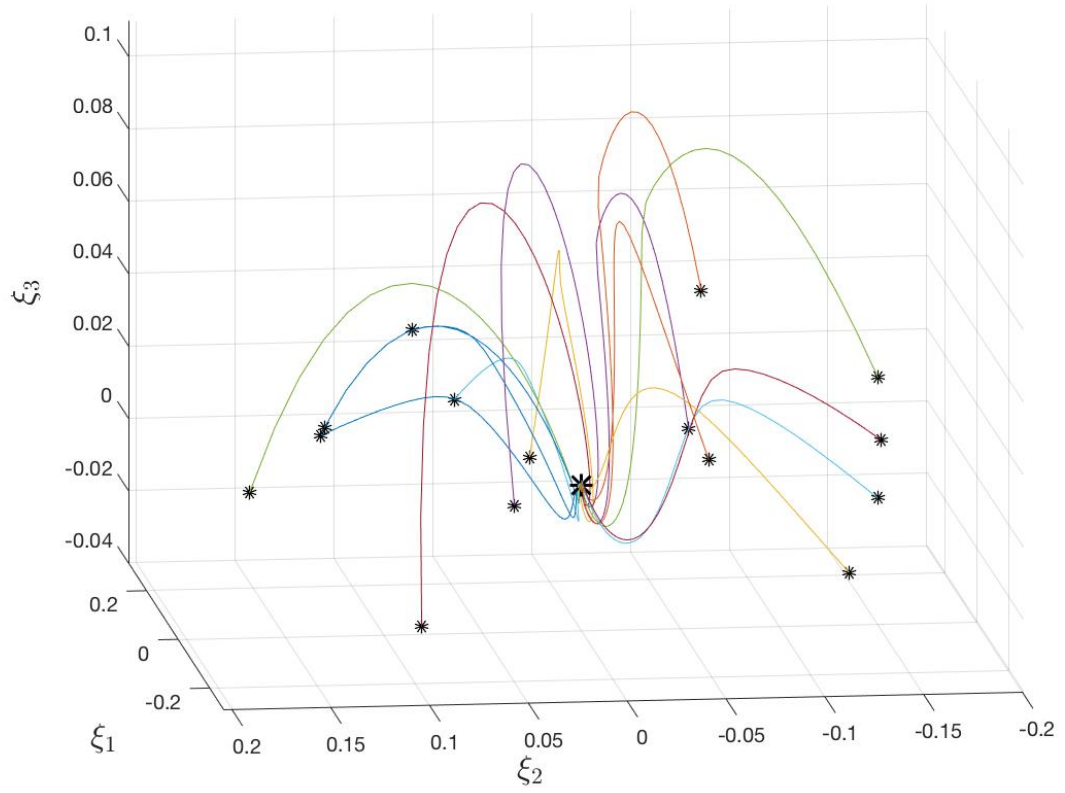


Figure 7: Plot of the various trajectories generated by SEDS when using only two gaussians evaluated with MSE.

Although increasing the number of gaussians systematically decreases the error, there is an optimal number of gaussians. This optimality is given largely by the degradation of performance of the model due to overfitting and to some extent by the added computational costs.

We firstly look at overfitting, which can be clearly visible with K=6 gaussians in Figure 8. We can see in two cases that a single gaussian is used to describe 2 very similar trajectories, this has an effect that the error for these trajectories will be very low, but a trajectory that is slightly more dissimilar will be almost impossible to model. As an example, the trajectory represented by the red arrow will be very difficult to model effectively. The extreme case of this overfitting phenomenon would be with K=15 gaussians where each training trajectory would have its associated gaussian. It is also important to notice that in the values of Table 1 we are only measuring the errors of the training trajectories. Additional testing trajectories would likely display a number of gaussians where the model is ideal.

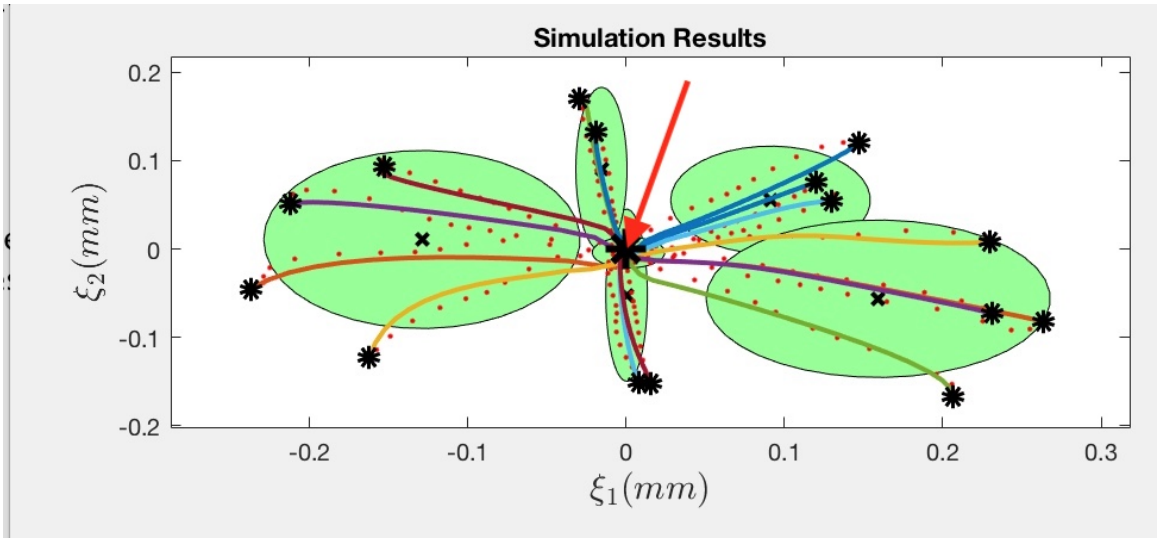
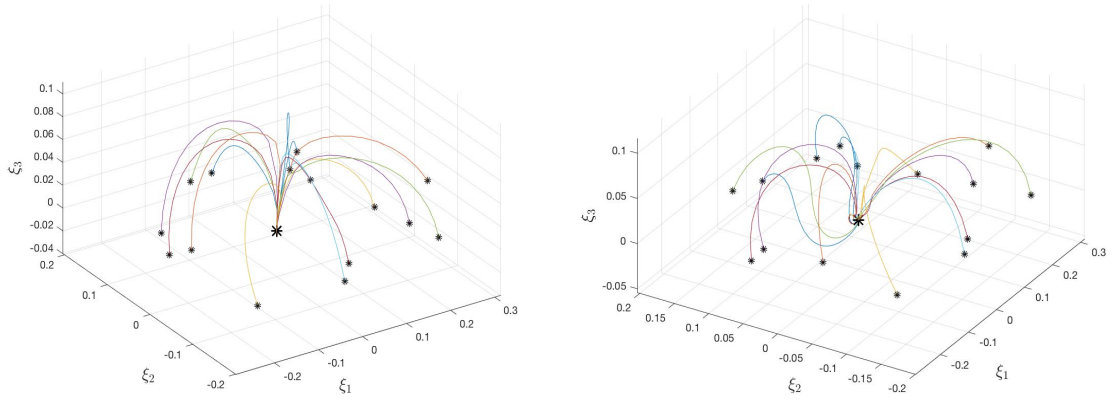


Figure 8: Plot of the various trajectories generated by SEDS when using 6 gaussians, causing overfitting.

Furthermore, increasing the number of gaussians implies a significant increase in computational costs. We can account for this by using a hybrid measure such as AIC or BIC which takes into account the computational complexity. Using this and some additional testing data, one can find an optimal number of gaussians.

We finally made the decision to retain K=5 gaussians using more intuitive methods, firstly looking at the rate of change of the error measures: K=5 and K=6 gaussians gave similar errors. Secondly, we looked for a good graphical model that guarantees trajectories coming from positive ξ_3. A strong graphical result is given in Figure 9a.

Lastly, we look at the impact of the optimality criterion; MSE or Likelihood. In a qualitative sense, one can see immediately that the MSE criterion produces better results. This is visible in Figure 9b where the trajectories are far from being ideal in the case of the Likelihood criterion compared to the MSE result in Figure 9a, having the same number of gaussians.



(a) (b)

Figure 9: Plots of the various trajectories generated by SEDS when using 5 gaussians evaluated with (a) MSE, (b) the Likelihood function.

4.2 Exercise 4

We continue by evaluating the generated model. We save the trajectories saved by the robot for a number of different displacements. Plotting these trajectories on MATLAB (and centering them) enables us to compare the demonstrated trajectories with the produced trajectories. We can see that the two are relatively similar. We see, in particular that the trajectory rises adequately, and then descends onto the target position. This rise and fall behaviour is, however, less pronounced for shorter displacements as is visible by the green trajectory in Figure 10. This can potentially cause problems by toppling the tower if moving an object a very short distance. We also notice a similarity in that the trajectory smoothly moves away from the starting point, but then comes down vertically on the ending point. This is very good for a placing motion. For a picking motion it could be more problematic as the tool-piece could topple the tower in moving away. This is a primary disadvantage of only using a single model for both types of motion.

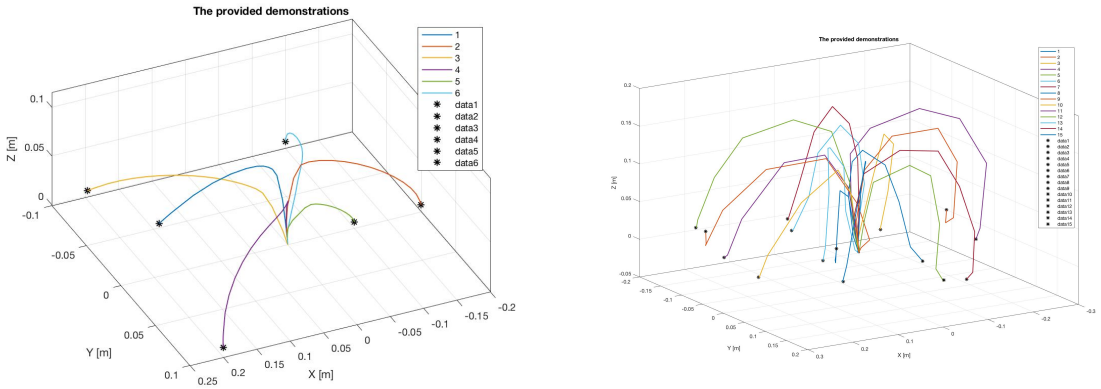


Figure 10: Left : MATLAB plots of real robot trajectories. Right : Matlab plots of demonstrated trajectories.

We further use the initial and final positions of each of the 6 robot trajectories to test the SEDS model. Using a modification of the MATLAB script "Simulate the model" we plot both the results of the model (in thin lines) and the measured robot trajectory (in thick lines), the result is visible on Figure 11. We observe that the simulated trajectories are firstly smoother and systematically (except for the light-blue trajectory) rise higher than the real trajectories. This could be for a number of reasons but is most likely attributed to the constraints in solving the inverse kinematics problem. However, in the general sense, the trajectories are very similar.

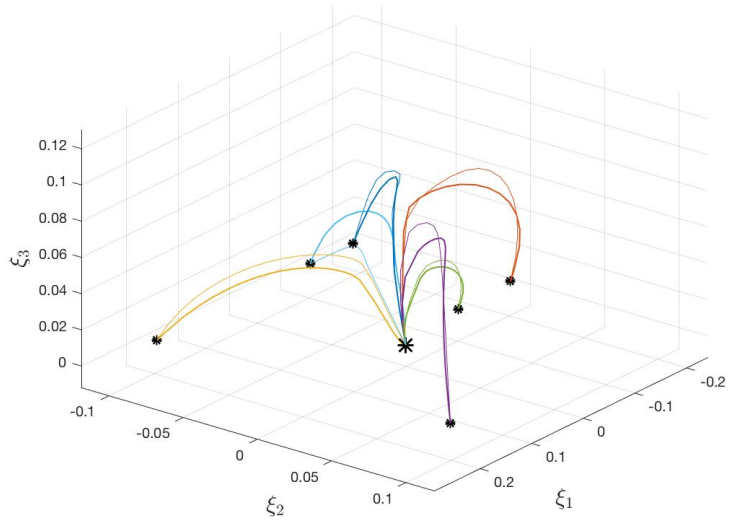


Figure 11: Comparison between the real robot trajectory (thick line) and the calculated MATLAB trajectory (fine line).

5 Building a tower game

In this final section we will describe how all the previous tasks come together in order to allow the robot to autonomously build a stable and relatively high tower. In particular, we will see how PCA and SEDS are used in this task and how we designed a stacking strategy in order to get the final tower according to our wishes, that is, object order, heights and any other specification.

5.1 Exercise 5

Before applying the chosen strategy, the robot must first approach the objects detected by the fixed camera and then use its wrist camera in order to perform PCA on each of them, as defined in the *Object recognition* section. This will allow the robot to identify the objects and to assign them to a value which will be used for the order of stacking. However, it is important to note that the distance between the camera and the end effector of the robot as well as the fact that position of the objects are given by both cameras led to inaccuracies in the positioning of the robot over a given object. Therefore, in order to correct for this variation, we chose to compute the polar coordinates of the robot's end effector (i.e. amplitude and angle) to be able to add a small offset in the distance from the center of the robot to the object, instead of going through the more time consuming process of correcting x-y values.

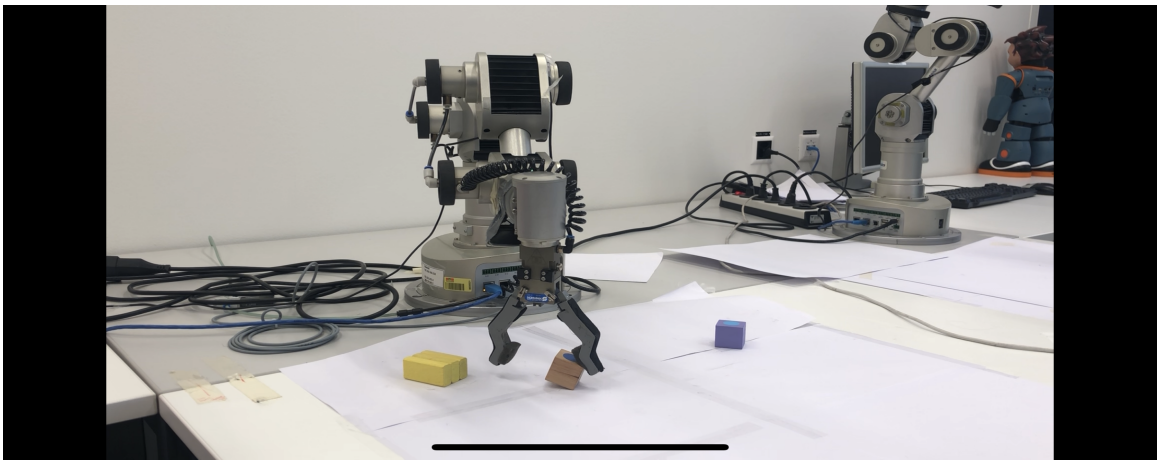
The object selection essentially defined the stacking strategy. Our set consisting of two small objects and one large object, we figured it would be best to have a simple and safe strategy which ensured stability of the tower, meaning we wished the robot to stack the objects as follows: small object number 1 $\rightarrow$ larger object $\rightarrow$ small object number 2. Since these objects have relatively the same general shape (parallelepiped), their height were pretty similar and therefore we chose the heights at which the robot would let go of the objects to be, by object order: $1.5cm$, $2cm$, *and* $2.5cm$, again to ensure overall tower stability. Figure 14 represents the chosen strategy of stacking.

Lastly, we will describe our results with a few situations we had. Although in the end we managed to get a tower according to our wishes and strategy, it did take some time and a few problem encounters which we will now get into.

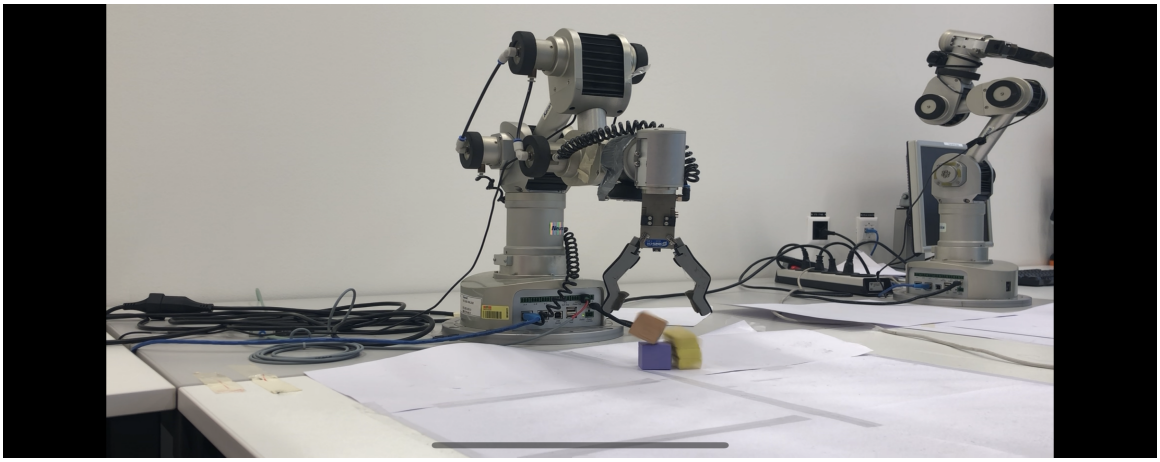
Unfortunately, due to circumstances out of our control, although the PCA algorithm worked quite well in the previous sections, here the robot was often unable to perform PCA on our objects, because of poor camera functioning, which led us to perform lots of trials and to eventually having to enter the positions of the objects manually with the sole purpose of observing a correct behaviour of the robot building the tower. Figure 12 shows erroneous behaviours of the robot due to poor PCA performance.

Moreover, the trajectories that we built in the previous section and that are now used by the robot could definitely be improved, namely during the picking phase. The robot generally did not go high enough after having just placed an object in order to move to the next one. This led to occasional collision with the end effector of the robot and the object that were in the, or even with the tower itself. Figure 13 shows such behaviour of the robot.

Lastly, a supplementary video of the robot performing the building of the tower is given in a separate file.

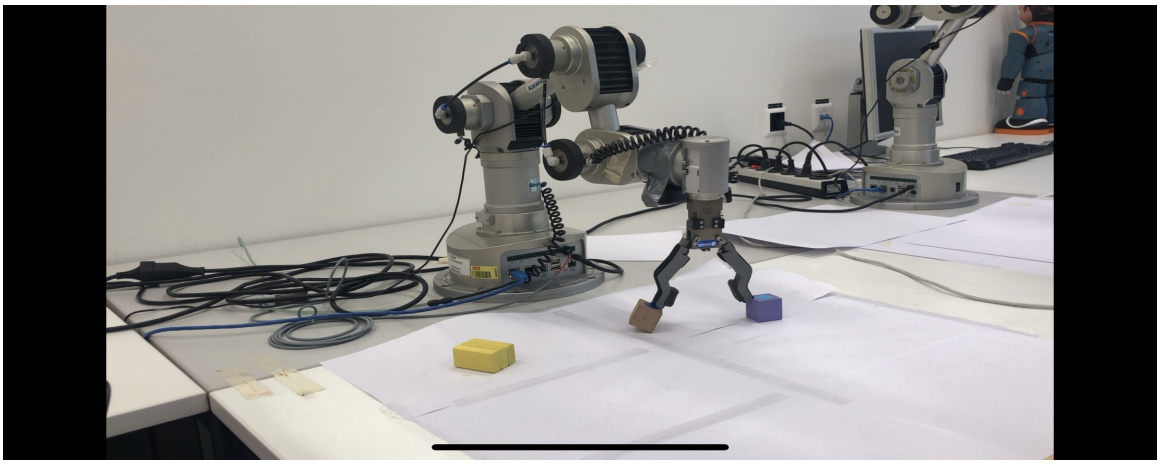


(a)

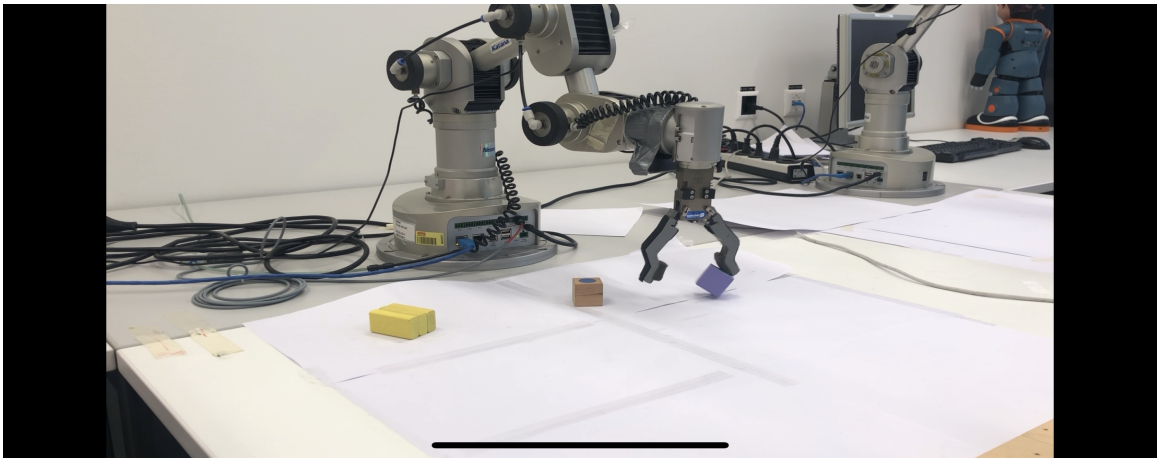


(b)

Figure 12: Pictures showing erroneous behaviour of the robot due to error in position determination when previously performing PCA. (a) Robot stomps on the object by missing its position. (b) Robot misses tower, therefore failed to put object on top of the previous one and destroys tower.



(a)



(b)

Figure 13: Pictures showing erroneous behaviour of the robot due to error in trajectory determination and execution when picking object - the arm does not go high enough to avoid objects in the way before executing picking. (a) Robot collides with most recently placed object and with next object. (b) Robot misses object to pick.

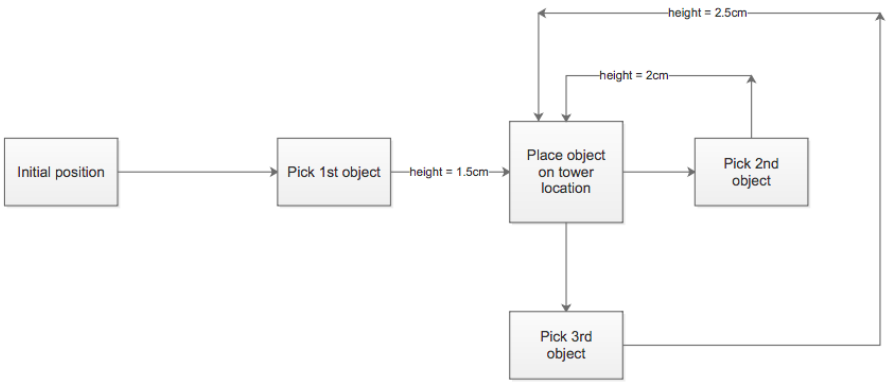


Figure 14: Block diagram representing the chosen block stacking strategy. The order of picking is defined as follows: 1st object → 2nd object → 3rd object.

6 Conclusion

We note that we ultimately managed to build a three block tower using the Katana robot. The errors with classification are unfortunate, especially given the strong classification results in the preparatory stages. This could be improved by looking at the eigenvectors in detail and perhaps including more or less to be able to reliably discern the three different objects we choose. Running the classification algorithm with completely different objects might also be the solution to getting a sufficiently robust performance. Unfortunately we lacked the time to dive deeply into this problem and find a solution. The trained trajectories ultimately were very good, especially after the adjustment in the polar reference frame. A significant improvement could be made by creating a motion model that is separate for the picking and placing motions. The strategy used to build the tower is simple but sufficiently effective for simple block shapes. An interesting extension of functionality would be to identify not a specific object but the type of object (square, triangle, rounded) and have the algorithm define a strategy accordingly.